Practitioner's Docket No. NAI1P481/01.207.01 *PATENT*

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Neil Andrew Cowie et al.

Application No.: 10/003,322 Group No.: 2131
Filed: 12/06/2001 Examiner: Syed Zia
For: INITIATING EXECUTION OF A COMPUTER PROGRAM FROM AN ENCRYPTED VERSION OF A COMPUTER PROGRAM

Mail Stop Appeal Briefs – Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF
(PATENT APPLICATION--37 C.F.R. § 41.37)

1. This brief is in furtherance of the Notice of Appeal, filed in this case on 11/30/2006, and in response to the Notice of Panel Decision from Pre-Appeal Brief Review, mailed 12/20/2006.

2. STATUS OF APPLICANT

This application is on behalf of other than a small entity.

3. FEE FOR FILING APPEAL BRIEF

Pursuant to 37 C.F.R. § 41.20(b)(2), the fee for filing the Appeal Brief is:

other than a small entity	$500.00
Appeal Brief fee due	**$500.00**

4. EXTENSION OF TERM

The proceedings herein are for a patent application and the provisions of 37 C.F.R. § 1.136 apply.

5. TOTAL FEE DUE

The total fee due is:

Appeal brief fee	$500.00
Extension fee (if any)	$0.00
TOTAL FEE DUE	**$500.00**

6. FEE PAYMENT

Authorization is hereby made to charge the amount of $500.00 to Deposit Account No. 50-1351(Order No.NAI1P481).

7. FEE DEFICIENCY

If any additional extension and/or fee is required, and if any additional fee for claims is required, charge Deposit Account No. 50-1351(Order No.NAI1P481).

Date: January 30, 2007

/KEVINZILKA/

Signature of Practitioner

Reg. No.: 41,429
Tel. No.: 408-971-2573
Customer No.: 28875

Kevin J. Zilka
Zilka-Kotab, PC
P.O. Box 721120
San Jose, CA 95172

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	)
	)
Cowie et al.	) Group Art Unit: 2131
	)
Application No. 10/003,322	) Examiner: ZIA, SYED
	)
Filed: 12/06/2001	) Date: January 30, 2007
	)
For: INITIATING EXECUTION OF A COMPUTER PROGRAM FROM AN ENCRYPTED VERSION OF A COMPUTER PROGRAM	)

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

ATTENTION: Board of Patent Appeals and Interferences

APPEAL BRIEF (37 C.F.R. § 41.37)

This brief is in furtherance of the Notice of Appeal, filed in this case on 11/30/2006, and in response to the Notice of Panel Decision from Pre-Appeal Brief Review, mailed 12/20/2006.

The fees required under § 1.17, and any required petition for extension of time for filing this brief and fees therefor, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

This brief contains these items under the following headings, and in the order set forth below (37 C.F.R. § 41.37(c)(i)):

I REAL PARTY IN INTEREST
II RELATED APPEALS AND INTERFERENCES
III STATUS OF CLAIMS
IV STATUS OF AMENDMENTS
V SUMMARY OF CLAIMED SUBJECT MATTER

VI GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

VII ARGUMENT

VIII CLAIMS APPENDIX

IX EVIDENCE APPENDIX

X RELATED PROCEEDING APPENDIX

The final page of this brief bears the practitioner's signature.

I REAL PARTY IN INTEREST (37 C.F.R. § 41.37(c)(1)(i))

The real party in interest in this appeal is McAfee, Inc.

II RELATED APPEALS AND INTERFERENCES (37 C.F.R. § 41.37(c) (1)(ii))

With respect to other prior or pending appeals, interferences, or related judicial proceedings that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no other such appeals, interferences, or related judicial proceedings.

A Related Proceedings Appendix is appended hereto.

III STATUS OF CLAIMS (37 C.F.R. § 41.37(c) (1)(iii))

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-4, 9-16, 21-28, and 33-39

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims withdrawn from consideration: None
2. Claims pending: 1-4, 9-16, 21-28, and 33-39
3. Claims allowed: None
4. Claims rejected: 1-4, 9-16, 21-28, and 33-39
5. Claims cancelled: 5-8, 17-20, and 29-32

C. CLAIMS ON APPEAL

The claims on appeal are: 1-4, 9-16, 21-28, and 33-39

See additional status information in the Appendix of Claims.

IV STATUS OF AMENDMENTS (37 C.F.R. § 41.37(c)(1)(iv))

As to the status of any amendment filed subsequent to final rejection, there are no such amendments after final.

V SUMMARY OF CLAIMED SUBJECT MATTER (37 C.F.R. § 41.37(c)(1)(v))

With respect to a summary of Claim 1, as shown in Figure 1, a computer program product for controlling a computer to execute a computer program within a computer memory is provided. The computer program product comprises a loader program (e.g. see item 2 of Figure 1, etc.) and an encrypted version of the computer program (e.g. see item 6 of Figure 1, etc.). The loader program (e.g. see item 2 of Figure 1, etc.) is operable to read the encrypted version of the computer program (e.g. see item 6 of Figure 1, etc.) stored in a program store, decrypt the encrypted version of the computer program (e.g. see item 6 of Figure 1, etc.) to form the computer program in an executable form (e.g. see item 9 of Figure 1, etc.), load the computer program (e.g. see item 9 of Figure 1, etc.) directly into the computer memory (e.g. see item 8 of Figure 1, etc.), and trigger execution of the computer program (e.g. see item 9 of Figure 1, etc.) as loaded into the computer memory (e.g. see item 8 of Figure 1, etc.) by the loader program (e.g. see item 2 of Figure 1, etc.). The computer program (e.g. see item 9 of Figure 1, etc.) that is decrypted, loaded, and executed includes a malware scanning computer program. Additionally, the malware scanning computer program is operable such that once executed, the malware scanning computer program scans the loader program (e.g. see item 2 of Figure 1, etc.) for malware. If the loader program (e.g. see item 2 of Figure 1, etc.) is detected as being infected with malware, then the malware scanning computer program is operable to repair the loader program (e.g. see item 2 of Figure 1, etc.) or replace the loader program (e.g. see item 2 of Figure 1, etc.) with a clean copy of the loader program (e.g. see item 2 of Figure 1, etc.). The malware scanning computer program is operable to scan for malware including one or more of a computer virus, a worm, a Trojan, a banned computer file, a banned word and a banned image. See, for example, page 3, lines 10-22; page 4, lines 26-32; page 5, lines 1-6; and page 5, lines 8-11 et al.

With respect to a summary of Claim 13, as shown in Figure 1, a method of executing of a computer program within a computer memory is provided. In use, a loader program (e.g. see item 2 of Figure 1, etc.) is executed. The loader program (e.g. see item 2 of Figure 1, etc.) operates to read an encrypted version of the computer program (e.g. see item 6 of Figure 1, etc.) stored in a program store, decrypt the encrypted version of the computer program (e.g. see item 6 of Figure 1, etc.) to form the computer program in an executable form (e.g. see item 9 of Figure

1, etc.), load the computer program (e.g. see item 9 of Figure 1, etc.) directly into the computer memory (e.g. see item 8 of Figure 1, etc.), and trigger execution of the computer program (e.g. see item 9 of Figure 1, etc.). In addition, the computer program (e.g. see item 9 of Figure 1, etc.) is executed as loaded into the computer memory (e.g. see item 8 of Figure 1, etc.) by the loader program (e.g. see item 2 of Figure 1, etc.). The computer program (e.g. see item 9 of Figure 1, etc.) that is decrypted, loaded, and executed includes a malware scanning computer program. Additionally, the malware scanning computer program is operable such that once executed, the malware scanning computer program scans the loader program (e.g. see item 2 of Figure 1, etc.) for malware. If the loader program (e.g. see item 2 of Figure 1, etc.) is detected as being infected with the malware, then the malware scanning computer program is operable to repair the loader program (e.g. see item 2 of Figure 1, etc.) or replace the loader program (e.g. see item 2 of Figure 1, etc.) with a clean copy of the loader program (e.g. see item 2 of Figure 1, etc.). Further, the malware scanning computer program is operable to scan for the malware including one or more of a computer virus, a worm, a Trojan, a banned computer file, a banned word and a banned image. See, for example, page 3, lines 10-22; page 4, lines 26-32; page 5, lines 1-6; and page 5, lines 8-11 et al.

With respect to a summary of Claim 25, as shown in Figure 1, an apparatus for executing a computer program within a computer memory is provided. The apparatus comprises loader program logic (e.g. see item 2 of Figure 1, etc.) and a program store operable to store an encrypted version of the computer program (e.g. see item 6 of Figure 1, etc.). The loader program logic (e.g. see item 2 of Figure 1, etc.) is operable to read the encrypted version of the computer program (e.g. see item 6 of Figure 1, etc.) stored in the program store, decrypt the encrypted version of the computer program (e.g. see item 6 of Figure 1, etc.) to form the computer program in an executable form (e.g. see item 9 of Figure 1, etc.), load the computer program (e.g. see item 9 of Figure 1, etc.) directly into the computer memory (e.g. see item 8 of Figure 1, etc.), and trigger execution of the computer program (e.g. see item 9 of Figure 1, etc.) as loaded into the computer memory (e.g. see item 8 of Figure 1, etc.) by the loader program (e.g. see item 2 of Figure 1, etc.). Additionally, the computer program that is decrypted, loaded, and executed includes a malware scanning computer program. The malware scanning computer program is operable such that once executed, the malware scanning computer program scans the loader program (e.g. see item 2 of Figure 1, etc.) for malware. If the loader program (e.g. see

item 2 of Figure 1, etc.) is detected as being infected with the malware, then the malware scanning computer program is operable to repair the loader program (e.g. see item 2 of Figure 1, etc.) or replace the loader program (e.g. see item 2 of Figure 1, etc.) with a clean copy of the loader program (e.g. see item 2 of Figure 1, etc.). Further, the malware scanning computer program is operable to scan for the malware including one or more of a computer virus, a worm, a Trojan, a banned computer file, a banned word and a banned image. See, for example, page 3, lines 10-22; page 4, lines 26-32; page 5, lines 1-6; and page 5, lines 8-11 et al.

Of course, the above citations merely provide examples of various claim language and possibly other features, and thus should not be construed as limiting in any manner.

VI GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL (37 C.F.R. § 41.37(c)(1)(vi))

Following, under each issue listed, is a concise statement setting forth the corresponding ground of rejection.

Issue # 1: The Examiner has rejected Claims 1-4, 9-16, 21-28, and 33-39 under 35 U.S.C. 103(a) as being unpatentable over Abu-Husein (U.S. Patent No. 6,895,506), in view of Nachenberg (U.S. Patent No. 5,826,013).

VII ARGUMENT (37 C.F.R. § 41.37(c)(1)(vii))

The claims of the groups noted below do not stand or fall together. In the present section, appellant explains why the claims of each group are believed to be separately patentable.

Issue # 1:

The Examiner has rejected Claims 1-4, 9-16, 21-28, and 33-39 under 35 U.S.C. 103(a) as being unpatentable over Abu-Husein (U.S. Patent No. 6,895,506), in view of Nachenberg (U.S. Patent No. 5,826,013).

Group #1: Claims 1-4, 9, 12-16, 21, 24-28, 33, and 36

With respect to the independent claims, the Examiner has relied on the following excerpt from the Nachenberg reference to make a prior art showing of appellant's claimed technique "wherein said computer program that is decrypted, loaded, and executed includes a malware scanning computer program."

> "The present invention is a polymorphic anti-virus module or PAM (200) for detecting polymorphic viruses (150) using mutation-engine specific information for each known polymorphic virus rather than heuristic stopper and booster code sequences. **The PAM system (200) comprises a CPU emulator (210) for emulating the target program, a virus signature scanning module (250) for scanning decrypted virus code, and an emulation control module (220), including a static exclusion module (230) and a dynamic exclusion module (240), for determining how long each target file is emulated before it is scanned.** The emulation control module (220) also includes data (222) specific to each known polymorphic virus (150) and organized in a format that facilitates comparison with target files being tested for infection. This data (222) includes instruction/interrupt usage profiles (224) for the mutation engines (162) of the known polymorphic viruses (150), as well as size and target file types (226) for these viruses. The emulation control module (220) also includes a table (228) having an entry for each known polymorphic virus (150) which can be flagged when characteristics inconsistent with the polymorphic virus are detected." (Col. 3, lines 3-24 - emphasis added)

Appellant respectfully asserts that the excerpt from Nachenberg relied upon by the Examiner merely teaches that the "[t]he PAM system (200) comprises a CPU emulator (210) for emulating the target program, a virus signature scanning module (250) for scanning decrypted virus code,

and an emulation control module (220), including a static exclusion module (230) and a dynamic exclusion module (240), for determining how long each target file is emulated before it is scanned" (Col. 3, lines 6-12 – emphasis added). However, "emulating the target program" and "scanning decrypted virus code" (emphasis added), as in Nachenberg, does not disclose a technique "wherein said computer program that is decrypted, loaded, and executed includes a malware scanning computer program" (emphasis added) as claimed by appellant. Clearly, scanning decrypted virus code simply, as in Nachenberg, fails to even suggest a "malware scanning computer program" that is "decrypted, loaded, and executed" (emphasis added), in the manner as claimed by appellant.

Further, with respect to the independent claims, the Examiner has relied on the following excerpt from the Nachenberg reference to make a prior art showing of appellant's claimed technique "wherein said malware scanning computer program is operable such that once executed, said malware scanning computer program scans said loader program for malware."

> "In accordance with the present invention, **the static exclusion module (230) examines the gross characteristics of the target file for attributes that are inconsistent with the mutation engine specific data for known polymorphic viruses (150)**. These characteristics are the type of target file, the size of the target file's load image, the presence of certain instructions at the file entry point, and the distance between the file entry point and the end of the load image. The last characteristic is useful because most viruses append themselves to the files they infect. In some cases, the static exclusion module (230) allows certain target files to be identified as infected without any emulation.
>
> The dynamic exclusion module (240) examines the instruction/interrupt usage profiles (224) of each known polymorphic virus (150) as each instruction is fetched for emulation. The instruction/interrupt usage profiles (224) indicate which polymorphic viruses (150) employ mutation engines that do not use the fetched instruction in decryption loops they generate, and the emulation control module (220) flags these viruses. The emulation control module (220) continues until all mutation engines have been flagged or until a threshold number of instructions have been emulated. The flagging technique implemented by the dynamic exclusion module (240) determines when emulation has proceeded to a point where **at least some code from the decrypted static virus body (160) may be scanned** and substantially reduces the number of instructions emulated prior to scanning the remaining target files without resort to booster or stopper heuristics." (Col. 3, lines 25-53 - emphasis added)

Appellant respectfully asserts that the excerpt from Nachenberg relied upon by the Examiner merely discloses that "the static exclusion module (230) examines the gross characteristics of the target file for attributes that are inconsistent with the mutation engine specific data for known polymorphic viruses" (Col. 3, lines 25-28). In addition, Nachenberg discloses that "at least some code from the decrypted static virus body (160) may be scanned" (Col. 3, lines 49-50). Clearly, teaching the examination of the target file for attributes that are inconsistent with the mutation engine specific data for known polymorphic viruses and that some code of decrypted static virus body may be scanned, as in Nachenberg, clearly fails to even suggest a technique "wherein said malware scanning computer program is operable such that once executed, said malware scanning computer program scans said loader program for malware" (emphasis added), as claimed by appellant.

In the Office Action dated 09/06/2006, the Examiner argued that "[a]n emulation control module includes a static exclusion module, a dynamic exclusion module, instruction/interrupt usage profiles for the mutation engines of the known polymorphic viruses, size and target file types for the viruses, and a table having an entry for each known polymorphic virus" and that "[a]s a result, cited prior art does implement and teach a system and method that relates to scanning computer code for viruses and for providing results pertaining to the viruses found by scanning of the initiation of execution of a computer program using a mechanism that seeks to protect the computer program from malicious alteration" (see Page 3).

Appellant respectfully disagrees with the Examiner's arguments and asserts that Nachenberg merely discloses that "[t]he PAM system (200) comprises a CPU emulator (210) for emulating the target program, a virus signature scanning module (250) for scanning decrypted virus code, and an emulation control module (220), including a static exclusion module (230) and a dynamic exclusion module (240), for determining how long each target file is emulated before it is scanned" (Col. 3, lines 7-13 – emphasis added). Further, Nachenberg discloses that "[o]nce the emulation phase has been terminated, scanning can begin on decrypted static virus body 160 or at least those parts decrypted by the first 1.5 million instructions" (Col. 9, lines 50-52 – emphasis added).

However, merely disclosing scanning the decrypted static virus body once an emulation phase is terminated, as in Nachenberg, simply fails to even suggest a technique "wherein said malware scanning computer program is operable such that once executed, said malware scanning computer program scans said loader program for malware" (emphasis added), as claimed by appellant. Clearly, a system in which instructions of a target program are emulated before the target program is scanned, as in Nachenberg, fails to meet and even *teaches away* from a technique where "said malware scanning computer program is operable such that once executed, said malware scanning computer program scans said loader program for malware" (emphasis added), in the manner as claimed by appellant.

In addition, with respect to the independent claims, the Examiner has relied on the following excerpt from the Nachenberg reference to make a prior art showing of appellant's claimed technique "wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to repair said loader program or replace said loader program with a clean copy of said loader program."

> "Once the emulation phase has been terminated, scanning can begin on decrypted static virus body 160 or at least those parts decrypted by the first 1.5 million instructions. In order to facilitate scanning of static virus body 160, emulation control module 220 keeps track of which locations of virtual memory are most likely to be infected. In particular, emulation control module 220 tags the page or pages of virtual memory containing an instruction executed by emulator 210. In addition, **every time an emulated instruction writes to memory, the altered pages are tagged as containing modified data. Tagged pages in virtual memory are scanned for signatures of static virus body** 160 as follows:
>
> (1) if a page contains executed code, it and the following page are scanned
>
> (2) if a page has been written to during emulation and more than a threshold number of memory writes occurred anywhere in memory during emulation, the modified page and the following page are scanned." (Col. 9, lines 50-67 - emphasis added)

Appellant respectfully asserts that the excerpt from Nachenberg relied upon by the Examiner merely discloses that "every time an emulated instruction writes to memory, the altered pages are tagged as containing modified data." Further, Nachenberg teaches that "[t]agged pages in virtual memory are scanned for signatures of static virus body." However, merely scanning tagged pages in virtual memory, as in Nachenberg, fails to even suggest a technique "wherein, if said

loader program is detected as being infected with said malware, then said malware scanning computer program is operable to repair said loader program or replace said loader program with a clean copy of said loader program" (emphasis added), as claimed by appellant.

In the Office Action dated 09/06/2006, the Examiner argued that "[t]he computer virus detection module includes a CPU emulator for emulating a target program, and a virus signature scanning module for scanning decrypted virus code" (see Page 3). However, appellant respectfully asserts that merely "scanning [the] static virus body" and scanning "[t]agged pages in virtual memory...for signatures of [a] static virus body" (emphasis added), as in Nachenberg, does not even suggest a program "operable to repair said loader program or replace said loader program with a clean copy of said loader program" (emphasis added), in the manner as claimed by appellant. Further, appellant respectfully asserts that Nachenberg merely discloses that "[i]n order to return control of the computer to program code 120 following execution of virus 130, CS, IP, SS, and SP of uninfected image 100 are retained by virus 130" (Col. 5, lines 36-39 – emphasis added). Clearly, the virus retaining uninfected fields in order to return control after the execution of the virus, as in Nachenberg, simply fails to even suggest a program "operable to repair said loader program or replace said loader program with a clean copy of said loader program" (emphasis added), in the manner as claimed by appellant.

To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art and not based on appellant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed.Cir.1991).

Appellant respectfully asserts that at least the third element of the *prima facie* case of obviousness has not been met, since the prior art references, when combined, fail to teach or suggest all of the claim limitations, as noted above.

Group #2: Claims 10, 22, and 34

With respect to Claim 10 et al., the Examiner has relied on the following excerpt from the Abu-Husein reference to make a prior art showing of appellant's claimed technique "wherein said computer program is operable to terminate said loader program when said computer program is triggered to execute by said loader program" (see the same, or similar language, in each of the aforementioned claims).

> "Finally, it must be noted that, as well understood by those of ordinary skill in the relevant arts, the execution of a Process 16 in or by a PU 12 is conducted in an operating environment that is generally referred to as a "context". The context in which a Process 16 is executed includes, for example, the current operating state of the system, the current execution state of the Process 16 and the program the Process 16 is executing, and so on. In general, and typically, each Process 16 is executed in a corresponding context and the execution of a different Process 16 involves or requires a change of the system operating context to another context corresponding to the new Process 16. **Such changes in context, which are often referred to as "context switches" or "context swaps", may occur, for example, when a Process 16 must call another Process 16 for an operation or has been completed or when the processor "time slice" allocated to a Process 16 has ended and the processor is to be allocated to the execution of a different Process 16.**" (Col. 8, line 65 – Col. 9, line 15 – emphasis added)

Appellant respectfully asserts that the excerpt from Abu-Husein relied upon by the Examiner merely discloses that '"context switches" ... may occur ... when a Process 16 must call another Process 16 for an operation or has been completed or when the processor "time slice" allocated to a Process 16 has ended and the processor is to be allocated to the execution of a different Process 16' (Col. 9, lines 10-15 – emphasis added). Clearly, the concept of "time slices" and "context switches," (emphasis added), as in Abu-Husein, simply fails to even suggest a technique "wherein said computer program is operable to terminate said loader program when said computer program is triggered to execute by said loader program" (emphasis added), as claimed by appellant.

In the Office Action dated 09/06/2006, the Examiner failed to specifically respond to appellant's arguments and simply dismissed them as non-persuasive. Appellant emphasizes that Abu-Husein discloses that "[d]uring a context switch, the execution of the currently executing Process 16 is suspended and the storage space in Memory 12C is reassigned by the Memory Manager

30A to accommodate the programs and data of the new Process 16" (Col. 9, lines 16-19 – emphasis added). Clearly, Abu-Husein's disclosure that the currently executing process is suspended during a context switch, simply fails to even suggest that the "computer program is operable to terminate said loader program when said computer program is triggered to execute by said loader program" (emphasis added), in the manner as claimed by appellant.

Again, appellant respectfully asserts that at least the third element of the *prima facie* case of obviousness has not been met, since the prior art references, when combined, fail to teach or suggest all of the claim limitations, as noted above.

Group #3: Claims 11, 23, and 35

With respect to Claim 11 et al., the Examiner has relied on the following excerpt from the Abu-Husein reference to make a prior art showing of appellant's claimed technique "wherein said loader program is operable to load said computer program into a memory space within said computer memory separate from a memory space used by said loader program" (see the same, or similar language, in each of the aforementioned claims).

> "During a context switch, **the execution of the currently executing Process 16 is suspended and the storage space in Memory 12C is reassigned by the Memory Manager 30A to accommodate the programs and data of the new Process 16, thereby requiring that the context of the current Process 16 be saved to allow the execution of the Process 16 to be resumed at the point it was suspended when the Process 16 is "switched" or "swapped" in and again becomes the Process 16 currently being executed.** The saving and return of context state is typically performed by a Context Switch Mechanism 30C that operates co-operatively with the Memory Manager 30A and that maintains and manages a Context Memory 30D which may reside on Storage 12E or in Memory 12C and is used to store the essential context information, identified in FIG. 2 as Context State 30E, of a context being switched or swapped out of Memory 12C. As well understood by those of ordinary skill in the relevant arts, a Context Memory 30D may be implemented in a number of structures and formats and the context information may include, for example, the state of execution of a program being executed by the Process 16, data being generated as a result of the Process 16 operations, and so on.
>
> In the presently preferred embodiment of the present invention, and in those embodiments requiring provision for context switching, the Context Switch Mechanism 30C is modified to include at least the Header 18H of a Program 18 among the Context State 30E elements stored in Context Memory 30D when the Process 16 in which the Program 18 is being

> executed is context switched. Context Switch Mechanism 30C then returns the Header 18H to Memory 12C when the Process 16 in which the Program 18 is being executed is returned or restored for execution, and the context of the Process 18 is correspondingly returned, so that Header 18H is available to Load/Decrypt 28 during continuation of the execution of the Program 18." (Col. 9, lines 16-50 – emphasis added)

Appellant respectfully asserts that excerpt from Abu-Husein relied upon by the Examiner merely discloses a method of performing a "context switch." Specifically, Abu-Husein discloses a context switch where "the execution of the currently executing Process 16 is suspended and the storage space in Memory 12C is reassigned by the Memory Manager 30A to accommodate the programs and data of the new Process 16."

Additionally, Abu-Husein teaches that the context of the current process is saved 'to allow the execution of the Process 16 to be resumed at the point it was suspended when the Process 16 is "switched" or "swapped" in and again becomes the Process 16 currently being executed.' However, merely disclosing a context switch, as in Abu-Husein, fails to even suggest a technique "wherein said loader program is operable to load said computer program into a memory space within said computer memory separate from a memory space used by said loader program" (emphasis added), as claimed by appellant.

In the Office Action dated 09/06/2006, the Examiner failed to specifically respond to appellant's arguments and simply dismissed them as non-persuasive. Appellant emphasizes that Abu-Husein merely discloses that "Program Loader/Decryption Mechanism (Load/Decrypt) 28, which may be implemented, for example, as a program controlled Process 16 executed by Processor 12A" (Col. 6, lines 53-57 – emphasis added). Further, Abu-Husein discloses that "Load/Decrypt 28 is similar to a conventional loading utility in responding to calls, commands or requests for the execution of a program or a program module or function by operating in conjunction with a Memory Manager 30A to read the corresponding program code from Storage 12E and to write the program code into Memory 12C at locations selected and managed by Memory Manager 30A for execution by Processor 12A" (Col. 6, lines 60-67 – emphasis added).

However, the disclosure that the Load/Decrypt, which is a program controlled process executed by the Processor, operates in conjunction with Memory Manager to read the code from Storage and write the code into Memory for execution by the Processor, as in Abu-Husein, clearly fails to

even suggest a technique "wherein said loader program is operable to load said computer program into a memory space within said computer memory **separate** from a memory space used by said loader program" (emphasis added), as claimed by appellant.

Again, appellant respectfully asserts that at least the third element of the *prima facie* case of obviousness has not been met, since the prior art references, when combined, fail to teach or suggest all of the claim limitations, as noted above.

Group #4: Claim 37

With respect to dependent Claim 37, the Examiner has relied on Col. 3, lines 25-53 from the Nachenberg reference (reproduced above) to make a prior art showing of appellant's claimed technique "wherein, as a first task, said malware scanning computer program scans said loader program for said malware."

Appellant respectfully points out that the excerpt relied on by the Examiner merely teaches that "the static exclusion module (230) examines the gross characteristics of the target file for attributes that are inconsistent with the mutation engine specific data for known polymorphic viruses" (Col. 3, lines 25-28) and that "[t]he emulation control module (220) continues until all mutation engines have been flagged or until a threshold number of instructions have been emulated" (Col. 3, lines 44-46 – emphasis added). However, examining characteristics of a target file and emulating intrusions, as in Nachenberg, does not teach scanning a loader program, much less a specific technique "wherein, as a first task, said malware scanning computer program scans said loader program for said malware" (emphasis added), as claimed by appellant.

Again, appellant respectfully asserts that at least the third element of the *prima facie* case of obviousness has not been met, since the prior art references, when combined, fail to teach or suggest all of the claim limitations, as noted above.

Group #5: Claim 38

With respect to dependent Claim 38, the Examiner has relied on Col. 9, lines 50-67 (reproduced above), in addition to the excerpt below, from the Nachenberg reference to make a prior art showing of appellant's claimed technique "wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to replace said loader program with a clean copy of said loader program."

> "Upon infection by computer virus 130, header 110' of executable image 100' is modified so that size field equals the size of executable image 100 incremented by the size of computer virus 130. In addition, **computer virus 130 has replaced CS, IP of image 100 with CS', IP' in image 100'**. CS', IP' point to an entry point 132 of virus 130 rather than entry point 122 of program code 120. Similarly, computer virus 130 has replaced SS, SP of image 100 with SS', SP' in image 100', which point to the end of virus 130. **In order to return control of the computer to program code 120 following execution of virus 130, CS, IP, SS, and SP of uninfected image 100 are retained by virus 130.**" (Col. 5, lines 27-39 – emphasis added)

Appellant respectfully asserts that the excerpts from Nachenberg relied upon by the Examiner merely disclose that "every time an emulated instruction writes to memory, the altered pages are tagged as containing modified data" (Col. 9, lines 58-60) and that "[t]agged pages in virtual memory are scanned for signatures of static virus body" (Col. 9, lines 60-61). Further, Nachenberg discloses that "[u]pon infection by computer virus 130... computer virus 130 has replaced CS [code segment field], IP [instruction pointer field] of image 100 with CS', IP' in image 100'" (Col. 5, lines 27-31 – emphasis added) and that "[i]n order to return control of the computer to program code 120 following execution of virus 130, CS, IP, SS, and SP of uninfected image 100 are retained by virus 130" (Col. 5, lines 35-38 – emphasis added).

However, appellant notes that teaching the scanning of tagged pages in virtual memory, in addition to disclosing that a computer virus replaces code segment and instruction pointer fields, and retains the instruction pointers in order to return control to the computer program, as in Nachenberg, fails to teach a technique "wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to replace said **loader program** with a clean copy of said loader program" (emphasis added), as claimed by appellant. Clearly, the virus retaining instruction pointers to the uninfected image, as in Nachenberg, fails to meet "said malware scanning computer program is operable to replace said

loader program with a clean copy of said loader program" (emphasis added), in the manner as claimed by appellant.

Again, appellant respectfully asserts that at least the third element of the *prima facie* case of obviousness has not been met, since the prior art references, when combined, fail to teach or suggest all of the claim limitations, as noted above.

Group #6: Claim 39

With respect to dependent Claim 39, the Examiner has relied on Col. 9, lines 50-67, and Col. 5, lines 27-39 from the Nachenberg reference (reproduced above) to make a prior art showing of appellant's claimed technique "wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to repair said loader program."

Appellant respectfully asserts that the excerpts from Nachenberg relied upon by the Examiner merely disclose that "every time an emulated instruction writes to memory, the altered pages are tagged as containing modified data" (Col. 9, lines 58-60) and that "[t]agged pages in virtual memory are scanned for signatures of static virus body" (Col. 9, lines 60-61). Further, Nachenberg discloses that "[u]pon infection by computer virus 130... computer virus 130 has replaced CS [code segment field], IP [instruction pointer field] of image 100 with CS', IP' in image 100'" (Col. 5, lines 27-31 – emphasis added) and that "[i]n order to return control of the computer to program code 120 following execution of virus 130, CS, IP, SS, and SP of uninfected image 100 are retained by virus 130" (Col. 5, lines 35-38 – emphasis added).

However, appellant respectfully asserts that that teaching the scanning of tagged pages in virtual memory, in addition to disclosing that a computer virus replaces code segment and instruction pointer fields and retains the instruction pointers in order to return control to the computer program, as in Nachenberg, fails to teach a technique "wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to **repair** said loader program" (emphasis added), as claimed by appellant. Clearly, the virus replacing code and instruction pointers, and retaining the instruction pointers to the

uninfected image, as in Nachenberg, fails to suggest that "said malware scanning computer program is operable to **repair** said loader program" (emphasis added), in the manner as claimed by appellant.

Again, appellant respectfully asserts that at least the third element of the *prima facie* case of obviousness has not been met, since the prior art references, when combined, fail to teach or suggest all of the claim limitations, as noted above.

In view of the remarks set forth hereinabove, all of the independent claims are deemed allowable, along with any claims depending therefrom.

VIII CLAIMS APPENDIX (37 C.F.R. § 41.37(c)(1)(viii))

The text of the claims involved in the appeal (along with associated status information) is set forth below:

1. (Previously Presented) A computer program product for controlling a computer to execute a computer program within a computer memory, said computer program product comprising:

(a) a loader program; and

(b) an encrypted version of said computer program; wherein said loader program is operable to:

(i) read said encrypted version of said computer program stored in a program store;

(ii) decrypt said encrypted version of said computer program to form said computer program in an executable form;

(iii) load said computer program directly into said computer memory; and

(iv) trigger execution of said computer program as loaded into said computer memory by said loader program;

wherein said computer program that is decrypted, loaded, and executed includes a malware scanning computer program;

wherein said malware scanning computer program is operable such that once executed, said malware scanning computer program scans said loader program for malware;

wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to repair said loader program or replace said loader program with a clean copy of said loader program;

wherein said malware scanning computer program is operable to scan for said malware including one or more of a computer virus, a worm, a Trojan, a banned computer file, a banned word and a banned image.

2. (Original) A computer program product as claimed in claim 1, wherein said encrypted version of said computer program is encrypted with a private encryption key and said loader program is operable to decrypt said encrypted version of said computer program with a corresponding public key.

3. (Original) A computer program product as claimed in claim 1, wherein said encrypted version of said computer program and said loader program are stored as separate computer files within a computer file store.

4. (Original) A computer program product as claimed in claim 1, wherein said loader program is associated with initialisation data specifying one or more of:

a storage location of said encrypted version of said computer program;

a key to be used in decrypting said encrypted version of said computer program; and

parameters specifying how said computer program should be loaded into said computer memory for execution.

5. (Cancelled)

6. (Cancelled)

7. (Cancelled)

8. (Cancelled)

9. (Original) A computer program product as claimed in claim 1, wherein said loader program is operable to terminate after triggering execution of said computer program.

10. (Original) A computer program product as claimed in claim 1, wherein said computer program is operable to terminate said loader program when said computer program is triggered to execute by said loader program.

11. (Original) A computer program product as claimed in claim 1, wherein said loader program is operable to load said computer program into a memory space within said computer memory separate from a memory space used by said loader program.

12. (Original) A computer program product as claimed in claim 1, wherein said loader program is operable to load said computer program into an execution stream separate from an execution stream used by said loader program.

13. (Previously Presented) A method of executing of a computer program within a computer memory, said method comprising the steps of:

(a) executing a loader program, said loader program operating to:

(i) read an encrypted version of said computer program stored in a program store;

(ii) decrypt said encrypted version of said computer program to form said computer program in an executable form;

(iii) load said computer program directly into said computer memory; and

(iv) trigger execution of said computer program; and

(b) executing said computer program as loaded into said computer memory by said loader program;

wherein said computer program that is decrypted, loaded, and executed includes a malware scanning computer program;

wherein said malware scanning computer program is operable such that once executed, said malware scanning computer program scans said loader program for malware;

wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to repair said loader program or replace said loader program with a clean copy of said loader program;

wherein said malware scanning computer program is operable to scan for said malware including one or more of a computer virus, a worm, a Trojan, a banned computer file, a banned word and a banned image.

14. (Original) A method as claimed in claim 13, wherein said encrypted version of said computer program is encrypted with a private encryption key and said loader program decrypts said encrypted version of said computer program with a corresponding public key.

15. (Original) A method as claimed in claim 13, wherein said encrypted version of said computer program and said loader program are stored as separate computer files within a computer file store.

16. (Original) A method as claimed in claim 13, wherein said loader program is associated with initialisation data specifying one or more of:

a storage location of said encrypted version of said computer program;

a key to be used in decrypting said encrypted version of said computer program; and

parameters specifying how said computer program should be loaded into said computer memory for execution.

17. (Cancelled)

18. (Cancelled)

19. (Cancelled)

20. (Cancelled)

21. (Original) A method as claimed in claim 13, wherein said loader program terminates after triggering execution of said computer programs.

22. (Original) A method as claimed in claim 13, wherein said computer program terminates said loader program when said computer program is triggered to execute by said loader program.

23. (Original) A method as claimed in claim 13, wherein said loader program loads said computer program into a memory space within said computer memory separate from a memory space used by said loader program.

24. (Original) A method as claimed in claim 13, wherein said loader program loads said computer program into an execution stream separate from an execution stream used by said loader program.

25. (Previously Presented) Apparatus for executing a computer program within a computer memory, said apparatus comprising:

(a) loader program logic; and

(b) a program store operable to store an encrypted version of said computer program; wherein said loader program logic is operable to:

(i) read said encrypted version of said computer program stored in said program store;

(ii) decrypt said encrypted version of said computer program to form said computer program in an executable form;

(iii) load said computer program directly into said computer memory; and

(iv) trigger execution of said computer program as loaded into said computer memory by said loader program;

wherein said computer program that is decrypted, loaded, and executed includes a malware scanning computer program;

wherein said malware scanning computer program is operable such that once executed, said malware scanning computer program scans said loader program for malware;

wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to repair said loader program or replace said loader program with a clean copy of said loader program;

wherein said malware scanning computer program is operable to scan for said malware including one or more of a computer virus, a worm, a Trojan, a banned computer file, a banned word and a banned image.

26. (Original) Apparatus as claimed in claim 25, wherein said encrypted version of said computer program is encrypted with a private encryption key and said loader program logic is operable to decrypt said encrypted version of said computer program with a corresponding public key.

27. (Original) Apparatus as claimed in claim 25, wherein said encrypted version of said computer program and said loader program are stored as separate computer files within a computer file store.

28. (Original) Apparatus as claimed in claim 25, wherein said loader program logic is associated with initialisation data specifying one or more of:

a storage location of said encrypted version of said computer program;

a key to be used in decrypting said encrypted version of said computer program; and

parameters specifying how said computer program should be loaded into said computer memory for execution.

29. (Cancelled)

30. (Cancelled)

31. (Cancelled)

32. (Cancelled)

33. (Original) Apparatus as claimed in claim 25, wherein said loader program logic is operable to terminate after triggering execution of said computer programs.

34. (Original) Apparatus as claimed in claim 25, wherein said computer program logic is operable to terminate said loader program when said computer program is triggered to execute by said loader program.

35. (Original) Apparatus as claimed in claim 25, wherein said loader program logic is operable to load said computer program into a memory space within said computer memory separate from a memory space used by said loader program logic.

36. (Original) Apparatus as claimed in claim 25, wherein said loader program logic is operable to load said computer program into an execution stream separate from an execution stream used by said loader program logic.

37. (Previously Presented) A computer program product as claimed in claim 1, wherein, as a first task, said malware scanning computer program scans said loader program for said malware.

38. (Previously Presented) A computer program product as claimed in claim 1, wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to replace said loader program with a clean copy of said loader program.

39. (Previously Presented) A computer program product as claimed in claim 1, wherein, if said loader program is detected as being infected with said malware, then said malware scanning computer program is operable to repair said loader program.

IX EVIDENCE APPENDIX (37 C.F.R. § 41.37(c)(1)(ix))

There is no such evidence.

X RELATED PROCEEDING APPENDIX (37 C.F.R. § 41.37(c)(1)(x))

N/A

In the event a telephone conversation would expedite the prosecution of this application, the Examiner may reach the undersigned at (408) 971-2573. For payment of any additional fees due in connection with the filing of this paper, the Commissioner is authorized to charge such fees to Deposit Account No. 50-1351 (Order No. NAI1P481).

Respectfully submitted,

By: /KEVINZILKA/ Date: January 30, 2007

Kevin J. Zilka
Reg. No. 41,429

Zilka-Kotab, P.C.
P.O. Box 721120
San Jose, California 95172-1120
Telephone: (408) 971-2573
Facsimile: (408) 971-4660